Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

Introducción a la Programación
IIC1103

Prof. Ignacio Casas
icasas@ing.puc.cl

Tema 10 – Recursión Parte 3
Más Ejemplos

Colaboración de Valeria Herskovic

Recordar: recursividad

- Debe haber una salida no recursiva (caso base)
- Se invoca a sí mismo
  - Las llamadas recursivas deben converger al caso base, es decir, se debe ir reduciendo el tamaño del problema
- Cuidado con funciones recursivas infinitas
- Para ciertos problemas es más fácil implementar un método recursivo que uno iterativo (con ciclos).

Recordar: Tail recursion

- La llamada recursiva es la última instrucción y no está acompañada de otra sentencia.
- Una función recursiva normal potencialmente puede convertirse a *tail recursive* usando en la función original un parámetro adicional, usado para ir guardando un resultado de tal manera que la llamada recursiva ya no tiene una operación pendiente.

Ejemplo: suma recursiva

- suma(5) = 1+2+3+4+5

Ejemplo: suma recursiva

```
def suma(x):
 if x == 1:
  return x
 else:
  return x + suma(x - 1)
```

Tail recursion

```
def suma_tr(x, total=0):
  if x == 0:
    return total
  else:
    return suma_tr(x - 1, total + x)
```

Desafío: Ordenamiento Recursivo con el "Merge-Sort"

Supongamos que tenemos una lista **X** de números enteros desordenados. Definimos una función recursiva **mergesort(x,ip,iu)** que ordena la lista **X**.

X = lista inicialmente desordenada que queremos ordenar.

ip = índice inicio de la lista (valor inicial es **0**)

iu = último índice de la lista (valor inicial es **len(x)-1**)

Caso Base: la lista es de largo 1 o vacía; retorna sin hacer nada.

Caso Recursivo:

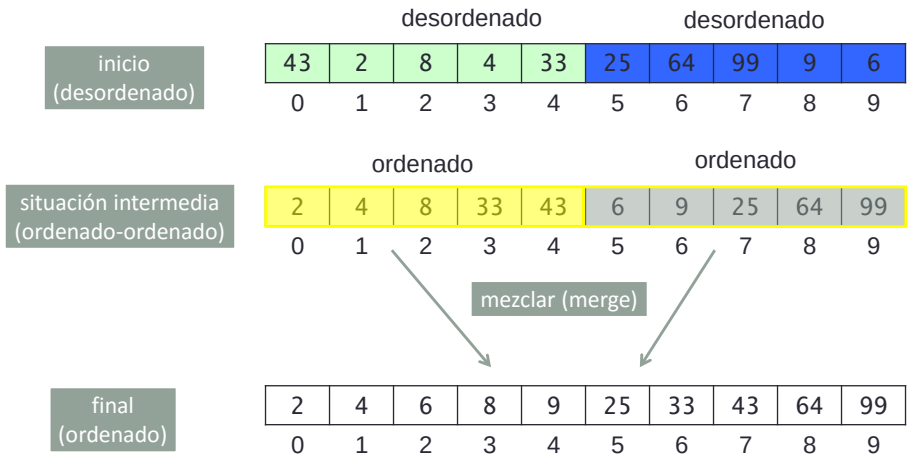
(1) La lista se divide (por la mitad) en dos partes

(2) Se invoca a si misma con la primera parte

(3) Se invoca a si misma con la segunda parte

(4) Mezcla las dos partes que ahora vienen ordenadas y retorna.

Ordenamiento Recursivo con Mergesort

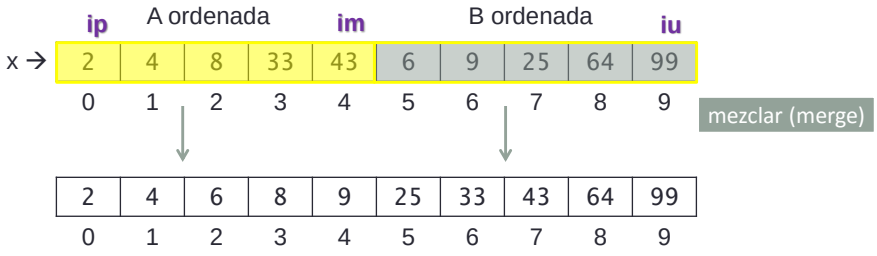


La "mezcla" en el Mergesort

Supongamos que ya tenemos una función para mezclar una lista **X** que contiene dos sub-listas ordenadas **A** y **B**:

```
def merge(x, ip, im, iu):
```

```
X  == la lista con dos sub-listas ordenadas A y B
ip == índice donde comienza sub-lista A
im == índice donde termina sub-lista A (im+1 indica el comienzo de B)
iu == índice donde termina sub-lista B
```



Solución Recursiva del Mergesort

Supongamos ahora que tenemos una lista desordenada:

43	2	8	4	33	25	64	99	9	6
0	1	2	3	4	5	6	7	8	9

Definimos un método recursivo: `def mergesort(x,ip,iu):`

¿Caso base? lista de tamaño 1 o 0 (ya está ordenada)

¿Caso recursivo? dividir la lista (+ o -) por la mitad
ordenar su 1a mitad y luego su 2a mitad,
Mezclar y retornar

Resultado final:

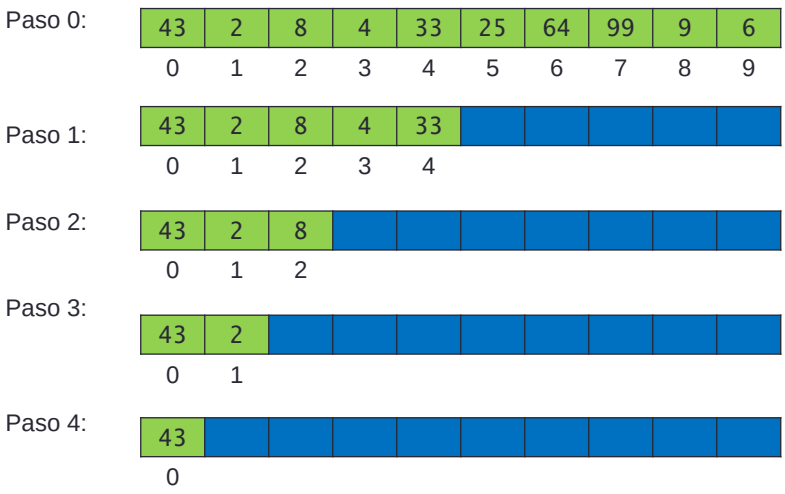
2	4	6	8	9	25	33	43	64	99
0	1	2	3	4	5	6	7	8	9

Mergesort Recursivo

```
def mergesort(x,ip,iu):
    if ip >= iu:
        return
    im = (ip+iu)//2
    mergesort(x,ip,im)    # ordenar 1a mitad
    mergesort(x,im+1,iu) # ordenar 2a mitad
    merge(x,ip,im,iu)
```

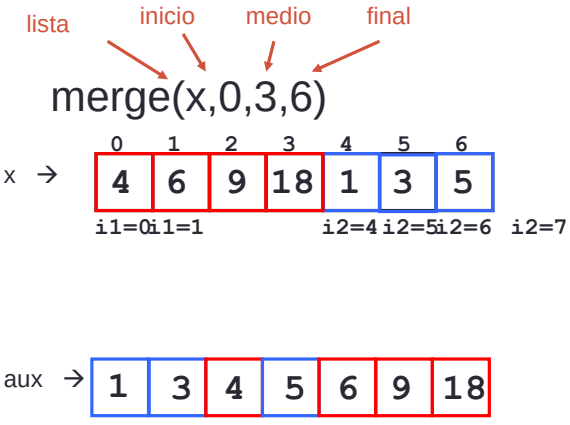
```
x  == la lista que dividimos en dos partes
ip == índice donde comienza sub-lista A (0 al comienzo)
im == índice donde termina sub-lista A
      (im+1 indica el comienzo de B)
iu == índice donde termina sub-lista B  (len(x)-1 al comienzo)
```

Solución Recursiva con el Mergesort



Retornar al Paso anterior y resolver la segunda parte de la lista. Luego mezclar. Y así sucesivamente.

La Mezcla (Merge) iterativa



Merge

función NO recursiva

```
def merge(x,ip,im,iu):
    aux=[]      # lista auxiliar
    i1 = ip     # indice  inicio 1a mitad
    i2 = im+1   # indice inicio 2a mitad
    while i1<=im and i2<=iu:   #hasta terminar 1 mitad
        if x[i1] < x[i2]:
            aux.append(x[i1])
            i1+=1
        else:
            aux.append(x[i2])
            i2+=1
    aux+= x[i1:im+1] # traspasar resto 1a mitad (si hay)
    aux+= x[i2:iu+1] # traspasar resto 2a mitad (si hay)
    x[ip:iu+1] = aux[0:len(aux)]  # copiar aux en x
```

Problema 2

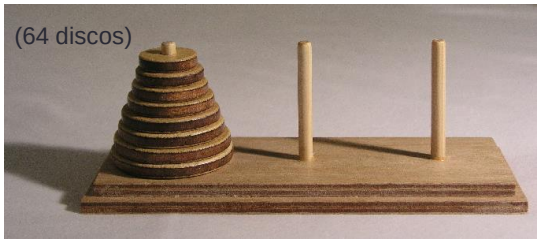
- Escriba un método recursivo que determine si una palabra es palíndrome o no.

Solución

```
def palindrome(x):
    if len(x)==0 or len(x)==1:
        return True
    if x[0]!=x[len(x)-1]:
        return False
    return palindrome(x[1:len(x)-1])
```

Problema 3: La leyenda de las Torres de Hanoi

Situación inicial

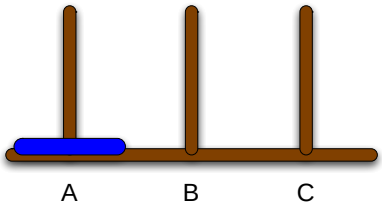


El mundo terminará cuando los monjes pasen 64 discos de una torre a la otra

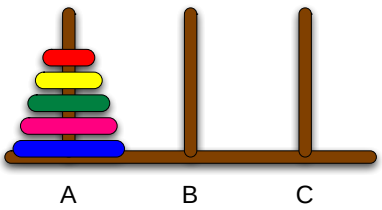
- Se deben mover todos los discos a otra estaca, con las siguientes reglas:
  1. Solo se puede mover un disco a la vez
  2. Cada movida consiste en sacar el disco que esté más arriba en una estaca y ponerlo en otra, arriba de los discos que ya estén.
  3. No se puede poner un disco sobre un disco más pequeño

Solución recursiva

- **Caso base (1 disco)**
  - Mover de A a C



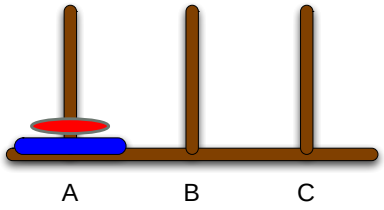
- **Caso recursivo (n discos)**
  - **Paso 1:** Mover n-1 discos de A a B
  - **Paso 2:** Mover 1 disco de A a C
  - **Paso 3.** Mover n-1 discos de B a C



http://www.dynamicdrive.com/dynamicindex12/towerhanoi.htm

Entendiendo la solución recursiva

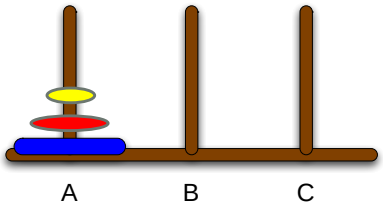
- Si tenemos 2 discos:
  1. Mover de A a B — **Paso 1**
  2. Mover de A a C — **Paso 2**
  3. Mover de B a C — **Paso 3**



- Si tenemos 3 discos:
  1. Mover de A a C
  2. Mover de A a B
  3. Mover de C a B

  Paso 1
  4. Mover de A a C — **Paso 2**
  5. Mover de B a A
  6. Mover de B a C

  Paso 3
  7. Mover de A a C



Solución recursiva

```
def Hanoi(n,Fuente,Auxiliar,Destino):
    if(n == 1):
        print ("disco en "+Fuente+" se mueve a "+Destino)
    else:
        Hanoi(n-1,Fuente,Destino,Auxiliar)
        print ("disco en "+Fuente+" se mueve a "+Destino)
        Hanoi(n-1,Auxiliar,Fuente,Destino)

# Main
print("solución para n = 2")
Hanoi(2,"A","B","C")
print()
print("solución para n = 3")
Hanoi(3,"A","B","C")
print()
print("solución para n = 4")
Hanoi(4,"A","B","C")
```

¿Cuánto tiempo toma la solución?

- Para n discos, toma 2^n-1 movidas.
- Para 64 discos, si los monjes se demoran 1 segundo por movida, demorarían 2^{64}-1 segundos, o:
- 585.000.000.000 años

Problema 4: Anagramas (permutaciones de letras) con algoritmo de Back-Tracking

¿Permutaciones de “hola” ?

hola
hoal
hloa
hlao
haol
halo

ohla
ohal
olha
olah
oahl
oalh

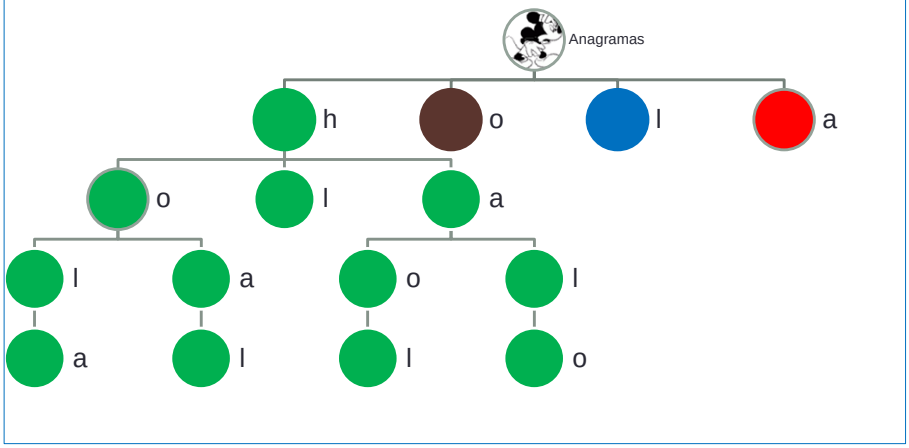
lhoa
lhao
loha
loah
laho
laoh

ahol
ahlo
aohl
aolh
alho
aloh

Caso base?
Caso recursivo?

Número de permutaciones: **n!**

Problema 4: visualización como árbol (incompleto)



Solución con **BackTracking**:
iterar para ir hacia el lado,
recursión para bajar por el árbol.

Tarea: completar este árbol con todos sus posibles caminos (para el lado y para abajo).

Problema 4: Anagramas (permutaciones de letras)

P("hola") función recursiva que *permuta* las 4 letras de "hola":
prefijo = ""
"h" + P("ola") "o" + P("hla") "l" + P("hoa") "a" + P("hol")

prefijo = prefijo + "h"
P("ola") función recursiva que *permuta* las 3 letras de "ola":
"o" + P("la") "l" + P("oa") "a" + P("ol")

prefijo = prefijo + "o" # prefijo == "ho"
P("la") función recursiva que *permuta* las 2 letras de "la":
"l" + P("a") "a" + P("l")

prefijo = prefijo + "l" # prefijo == "hol"
P("a") permuta "a" es el **Caso Base**:
imprime (prefijo + "a")

retornar a llamada anterior.

Problema 4: Anagramas (permutaciones de letras)

- $P(x_0x_1\ldots x_N) =$

$$\begin{cases} X_i + P(x_0, \ldots, x_{i-1}, x_{i+1}, \ldots, x_N) & \text{si } N>1 \\ X_N \quad \text{si } N=1 \end{cases}$$

- P (hola) =
  - h + P(ola)
  - o + P(hla)
  - l + P(hoa)
  - a + P(hol)

Anagramas

```
def anagramas(pal,prefijo=""):
    if len(pal) <= 1:
        print(prefijo+pal)
    else:
        for i in range(len(pal)):
            antes = pal[0:i]
            despues = pal[i+1:]
            anagramas(antes+despues,prefijo+pal[i])

anagramas("hola")
```

Desafío (muy difícil): No usar Iteración. (Hacer una recursión dentro de otra recursión)

Propuesto: usar diccionario para mostrar solo palabras válidas

Propuesto: permutaciones de listas

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

Introducción a la Programación
IIC1103

Prof. Ignacio Casas
icasas@ing.puc.cl

Tema 10 – Recursión Parte 3
Más Ejemplos

Colaboración de Valeria Herskovic